

2020 PERFORMANCE, PORTABILITY,
AND PRODUCTIVITY IN HPC FORUM



INVESTIGATION OF THE PERFORMANCE OF SYCL KERNELS ACROSS VARIOUS ARCHITECTURES



BRIAN HOMERDING

Leadership Computing Facility
Argonne National Laboratory
Speaker

OVERVIEW – SYCL ^[1]

- Cross-platform abstraction layer for heterogeneous programming
- Khronos standard specification
- Builds on the underlying concepts of OpenCL while including the strengths of single-source C++
- Includes hierarchical parallelism syntax and separation of data access from data storage
- Designed to be as close to standard C++ as possible

RAJA PERFORMANCE SUITE [2]

Collection of performance benchmarks with RAJA and non-RAJA variants.

- **Stream (stream)**

ADD, COPY, DOT, MUL, TRIAD

- **Basic (simple)**

DAXPY, IF_QUAD, INIT3, INIT_VIEW1D, INIT_VIEW1D_OFFSET, MULADDSUB, NESTED_INIT, REDUCE3_INT, TRAP_INT

- **LCALS (loop optimizations)**

DIFF_PREDICT, EOS, FIRST_DIFF, HYDRO_1D, HYDRO_2D, INT_PREDICT, PLANCKIAN

- **Apps (applications)**

DEL_DOT_VEC_2D, ENERGY, FIR, LTIMES, LTIMES_NOVIEW, PRESSURE, VOL3D

- **PolyBench (polyhedral optimizations)**

2MM, 3MM, ADI, ATAX, FDTD_2D, FLOYD_ARSHALL, GEMM, GEMVER, GESUMMV, HEAT_3D, JACOBI_1D, JACOBI_2D, MVT

RAJA PERFORMANCE SUITE

- Primary developer – Rich Hornung (LLNL)
 - See RAJAPerf github page for full list of contributors
- Very good for compiler testing
- Built in timer and correctness testing.
 - Timer cover full execution of many repetitions the kernels
 - Correctness is done with checksum compared against sequential execution
- Many “variants”
 - Base_Seq, Lambda_Seq, RAJA_Seq, Base_OpenMP, Lambda_OpenMP, RAJA_OpenMP, Base_OpenMPTarget, RAJA_OpenMPTarget, Base_CUDA, RAJA_CUDA

RAJA PERFORMANCE SUITE

- Primary developer – Rich Hornung (LLNL)
 - See RAJAPerf github page for full list of contributors
- Very good for compiler testing
- Built in timer and correctness testing.
 - Timer cover full execution of many repetitions the kernels
 - Correctness is done with checksum compared against sequential execution
- Many “Variants”
 - Base_Seq, Lambda_Seq, RAJA_Seq, Base_OpenMP, Lambda_OpenMP, RAJA_OpenMP, Base_OpenMPTarget, RAJA_OpenMPTarget, Base_CUDA, RAJA_CUDA, Base_SYCL

OUTLINE - P3

Lessons learned

- Productivity
 - Discuss experiences porting from CUDA
- Portability
 - Compiler correctness and support across various architectures
- Performance
 - Performance of various compilers for each architecture

PRODUCTIVITY



U.S. DEPARTMENT OF
ENERGY

Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.



PORTING FROM CUDA

- Memory Management
- Kernel Submission
- Kernel Code
- Argument Passing

Listing 1: CUDA Example

```
const size_t block_size = 256;

#define DATA_SETUP_CUDA \\\n    Double a; \\\n    cudaMalloc(a, iend); \\\n    cudaMemcpy(a, m_a, iend);

#define DATA_TEARDOWN_CUDA \\\n    cudaMemcpy(m_a, a, iend); \\\n    cudaFree(a);

__global__ void example(double a) {\n    size_t i = blockIdx.x * blockDim.x + threadIdx.x;\n    if (i < iend) {\n        EXAMPLE_BODY\n    }\n}

void EXAMPLE::runCudaVariant(VariantID vid) {\n    const size_t iend = getRunSize();\n    DATA_SETUP_CUDA;\n    startTimer();

    for (size_t irep = 0; irep , num_reps; ++irep) {\n        const size_t grid_size = DIVIDE_CEILING(iend,\n            block_size);\n        example<<<grid_size, block_size>>> (a, iend);\n    }

    stopTimer();\n    DATA_TEARDOWN_CUDA;\n}
```

Listing 2: SYCL Example

```
const size_t block_size = 256;

#define DATA_SETUP_SYCL \\\n    sycl::buffer<double> d_a {m_a, iend};

void EXAMPLE::runSyclVariant(VariantID vid) {\n    { // Buffer Scope\n        const size_t iend = getRunSize();\n        DATA_SETUP_SYCL;\n        startTimer();

        for (size_t irep = 0; irep , num_reps; ++irep) {\n            const size_t grid_size = block_size *\n                DIVIDE_CEILING(iend, block_size);\n            q.submit([&] (sycl::handler& h) {\n                auto a =\n                    d_a.get_access<sycl::access::mode::read_write>(h);\n\n                h.parallel_for<class EXAMPLE> (sycl::nd_range<1>\n                    {grid_size, block_size},\n                    [=] (sycl::nd_item<1> item) {\n\n                        size_t i = item.get_group(0) *\n                            item.get_local_range().get(0) +\n                            item.get_local_id(0);\n                        if (i < iend) {\n                            EXAMPLE_BODY\n                        }\n                    });\n            });\n        }\n    } // Buffer Destruction\n    stopTimer();\n}
```

PORTING FROM CUDA

- Memory Management
- Kernel Submission
- Kernel Code
- Argument Passing

Listing 1: CUDA Example

```
const size_t block_size = 256;

#define DATA_SETUP_CUDA \
    Double a; \
    cudaMalloc(a, iend); \
    cudaMemcpy(a, m_a, iend);

#define DATA_TEARDOWN_CUDA \
    cudaMemcpy(m_a, a, iend); \
    cudaFree(a);

__global__ void example(double a) {
    size_t i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < iend) {
        EXAMPLE_BODY
    }
}

void EXAMPLE::runCudaVariant(VariantID vid) {
    const size_t iend = getRunSize();
    DATA_SETUP_CUDA;
    startTimer();

    for (size_t irep = 0; irep , num_reps; ++irep) {
        const size_t grid_size = DIVIDE_CEILING(iend,
            block_size);
        example<<<grid_size, block_size>>> (a, iend);
    }

    stopTimer();
    DATA_TEARDOWN_CUDA;
}
```

Listing 2: SYCL Example

```
const size_t block_size = 256;

#define DATA_SETUP_SYCL \
    sycl::buffer<double> d_a {m_a, iend};

void EXAMPLE::runSyclVariant(VariantID vid) {
    { // Buffer Scope
        const size_t iend = getRunSize();
        DATA_SETUP_SYCL;
        startTimer();

        for (size_t irep = 0; irep , num_reps; ++irep) {
            const size_t grid_size = block_size *
                DIVIDE_CEILING(iend, block_size);
            q.submit([&] (sycl::handler& h) {
                auto a =
                    d_a.get_access<sycl::access::mode::read_write>(h);

                h.parallel_for<class EXAMPLE> (sycl::nd_range<1>
                    {grid_size, block_size},
                    [=] (sycl::nd_item<1> item) {

                        size_t i = item.get_group(0) *
                            item.get_local_range().get(0) +
                            item.get_local_id(0);
                        if (i < iend) {
                            EXAMPLE_BODY
                        }
                    });
            });

        } // Buffer Destruction
        stopTimer();
    }
}
```

PORTING FROM CUDA

- Memory Management
- Kernel Submission
- Kernel Code
- Argument Passing

Listing 1: CUDA Example

```
const size_t block_size = 256;

#define DATA_SETUP_CUDA \
    Double a; \
    cudaMalloc(a, iend); \
    cudaMemcpy(a, m_a, iend);

#define DATA_TEARDOWN_CUDA \
    cudaMemcpy(m_a, a, iend); \
    cudaFree(a);

__global__ void example(double a) {
    size_t i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < iend) {
        EXAMPLE_BODY
    }
}

void EXAMPLE::runCudaVariant(VariantID vid) {
    const size_t iend = getRunSize();
    DATA_SETUP_CUDA;
    startTimer();

    for (size_t irep = 0; irep , num_reps; ++irep) {
        const size_t grid_size = DIVIDE_CEILING(iend,
            block_size);
        example<<<grid_size, block_size>>>> (a, iend);
    }

    stopTimer();
    DATA_TEARDOWN_CUDA;
}
```

Listing 2: SYCL Example

```
const size_t block_size = 256;

#define DATA_SETUP_SYCL \
    sycl::buffer<double> d_a {m_a, iend};

void EXAMPLE::runSyclVariant(VariantID vid) {
    { // Buffer Scope
        const size_t iend = getRunSize();
        DATA_SETUP_SYCL;
        startTimer();

        for (size_t irep = 0; irep , num_reps; ++irep) {
            const size_t grid_size = block_size *
                DIVIDE_CEILING(iend, block_size);
            q.submit([&] (sycl::handler& h) {
                auto a =
                    d_a.get_access<sycl::access::mode::read_write>(h);

                h.parallel_for<class EXAMPLE> (sycl::nd_range<1>
                    {grid_size, block_size},
                    [=] (sycl::nd_item<1> item) {

                        size_t i = item.get_group(0) *
                            item.get_local_range().get(0) +
                            item.get_local_id(0);
                        if (i < iend) {
                            EXAMPLE_BODY
                        }
                    });
            });
        } // Buffer Destruction
        stopTimer();
    }
}
```

PORTING FROM CUDA

- Memory Management
- Kernel Submission
- Kernel Code
- Argument Passing

Listing 1: CUDA Example

```
const size_t block_size = 256;

#define DATA_SETUP_CUDA \\\n    Double a; \\\n    cudaMalloc(a, iend); \\\n    cudaMemcpy(a, m_a, iend);

#define DATA_TEARDOWN_CUDA \\\n    cudaMemcpy(m_a, a, iend); \\\n    cudaFree(a);

__global__ void example(double a) {\n    size_t i = blockIdx.x * blockDim.x + threadIdx.x;\n    if (i < iend) {\n        EXAMPLE_BODY\n    }\n}

void EXAMPLE::runCudaVariant(VariantID vid) {\n    const size_t iend = getRunSize();\n    DATA_SETUP_CUDA;\n    startTimer();

    for (size_t irep = 0; irep , num_reps; ++irep) {\n        const size_t grid_size = DIVIDE_CEILING(iend,\n            block_size);\n        example<<<grid_size, block_size>>>> (a, iend);\n    }

    stopTimer();\n    DATA_TEARDOWN_CUDA;\n}
```

Listing 2: SYCL Example

```
const size_t block_size = 256;

#define DATA_SETUP_SYCL \\\n    sycl::buffer<double> d_a {m_a, iend};

void EXAMPLE::runSyclVariant(VariantID vid) {\n    { // Buffer Scope\n        const size_t iend = getRunSize();\n        DATA_SETUP_SYCL;\n        startTimer();

        for (size_t irep = 0; irep , num_reps; ++irep) {\n            const size_t grid_size = block_size *\n                DIVIDE_CEILING(iend, block_size);\n            q.submit([&] (sycl::handler& h) {\n                auto a =\n                    d_a.get_access<sycl::access::mode::read_write>(h);\n\n                h.parallel_for<class EXAMPLE> (sycl::nd_range<1>\n                    {grid_size, block_size},\n                    [=] (sycl::nd_item<1> item) {\n\n                        size_t i = item.get_group(0) *\n                            item.get_local_range().get(0) +\n                            item.get_local_id(0);\n                        if (i < iend) {\n                            EXAMPLE_BODY\n                        }\n                    });\n            });\n        }\n    } // Buffer Destruction\n    stopTimer();\n}
```

PORTING FROM CUDA

- Memory Management
- Kernel Submission
- Kernel Code
- Argument Passing

Listing 1: CUDA Example

```
const size_t block_size = 256;

#define DATA_SETUP_CUDA \
    Double a; \
    cudaMalloc(a, iend); \
    cudaMemcpy(a, m_a, iend);

#define DATA_TEARDOWN_CUDA \
    cudaMemcpy(m_a, a, iend); \
    cudaFree(a);

__global__ void example(double a) {
    size_t i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < iend) {
        EXAMPLE_BODY
    }
}

void EXAMPLE::runCudaVariant(VariantID vid) {
    const size_t iend = getRunSize();
    DATA_SETUP_CUDA;
    startTimer();

    for (size_t irep = 0; irep , num_reps; ++irep) {
        const size_t grid_size = DIVIDE_CEILING(iend,
            block_size);
        example<<<grid_size, block_size>>> (a, iend);
    }

    stopTimer();
    DATA_TEARDOWN_CUDA;
}
```

Listing 2: SYCL Example

```
const size_t block_size = 256;

#define DATA_SETUP_SYCL \
    sycl::buffer<double> d_a {m_a, iend};

void EXAMPLE::runSyclVariant(VariantID vid) {
    // Buffer Scope
    const size_t iend = getRunSize();
    DATA_SETUP_SYCL;
    startTimer();

    for (size_t irep = 0; irep , num_reps; ++irep) {
        const size_t grid_size = block_size *
            DIVIDE_CEILING(iend, block_size);
        q.submit([&] (sycl::handler& h) {
            auto a =
                d_a.get_access<sycl::access::mode::read_write>(h);

            h.parallel_for<class EXAMPLE> (sycl::nd_range<1>
                {grid_size, block_size},
                [=] (sycl::nd_item<1> item) {

                size_t i = item.get_group(0) *
                    item.get_local_range().get(0) +
                    item.get_local_id(0);
                if (i < iend) {
                    EXAMPLE_BODY
                }
            });
        });
    }
    // Buffer Destruction
    stopTimer();
}
```

PORTABILITY



U.S. DEPARTMENT OF
ENERGY

Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.



SYCL ECOSYSTEM

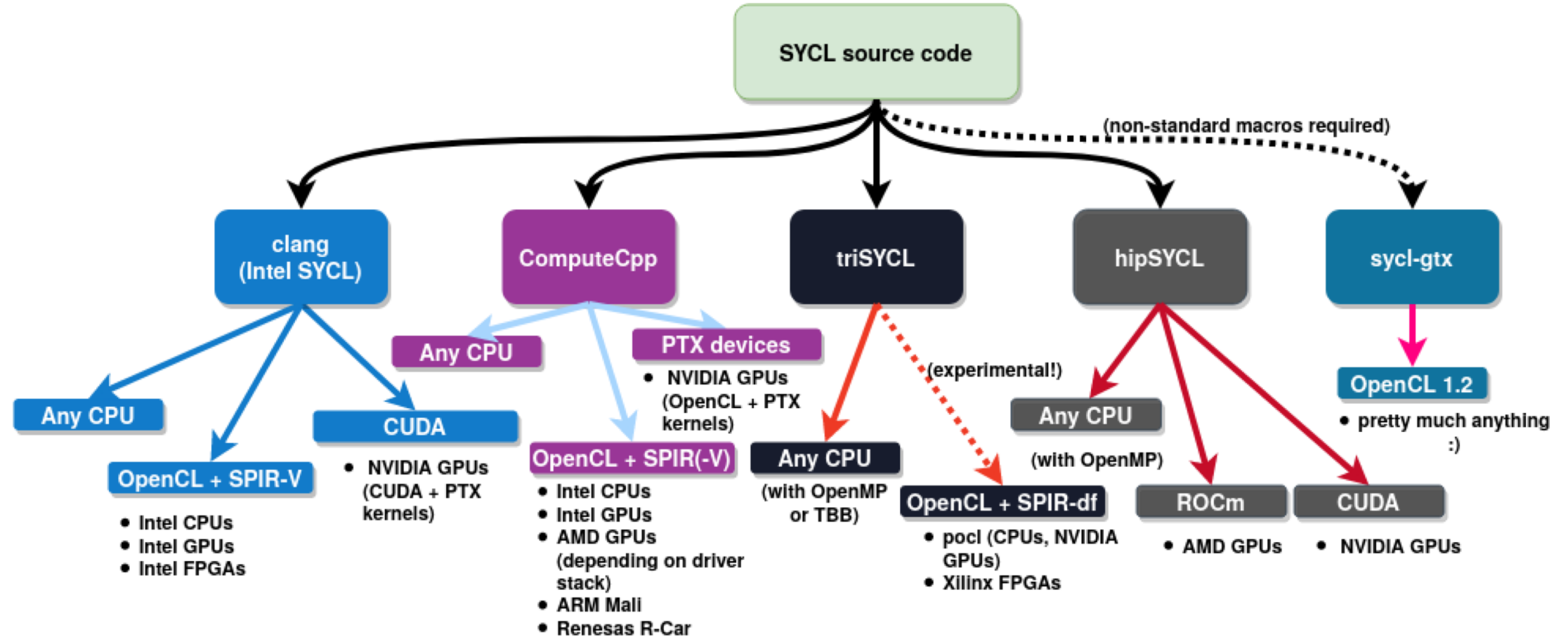


Image Credit [4]: <https://github.com/illuhad/hipSYCL/blob/develop/doc/img/sycl-targets.png>

COMPILERS

- Intel SYCL [3]
 - OpenCL + SPIRV for SKX and Gen9
 - CUDA + PTX for V100
- HipSYCL [4]
 - CUDA for V100
- ComputeCPP [5]
 - OpenCL + SPIRV for SKX and Gen9
 - OpenCL + PTX for V100

ARCHITECTURES

Measured performance [6]

- SKX – Intel Xeon Platinum Skylake 8180M Scalable processors
- Gen9 – Intel Xeon Processor E3-1585 v5, with Iris Pro Graphics P580
- V100 – NVIDIA V100 GPU

Processor	DP Flop-rate (GF/s)	DRAM (GB/s)
SKX	3,720	214
Gen9	300	28.8
V100	7,660	778

FEATURE SUPPORT

Workarounds for portability with current support

- Added extra boundary checks for kernels with buffers that are different size than the iteration space
- Syntactic sugar
 - `i.get_local_range(dim); -> i.get_local_range().get(dim);`
- Accessors with offset not fully supported, used pointer arithmetic
 - `auto x1 = d_x.get_access<read>(h, len, v1);`
– `-> auto x = d_x.get_access<read>(h);`
– `auto x1 = (x.get_pointer() + v1).get();`

FEATURE SUPPORT

Future support

- 3 Kernels with reductions are not included with our data
- Support is not standard for 1.2 specification
- 2020 specification additions of interests
 - Floating point atomics
 - Reductions
 - Unified shared memory
 - Lambda naming

CORRECTNESS

Checksum compared to sequential execution

- SKX - Intel SYCL
 - Several small floating point differences, within expected bounds
 - 1 incorrect result
- Gen9 - Intel SYCL
 - Several small floating point differences, within expected bounds
- V100 – HipSYCL
 - Several small floating point differences, within expected bounds
- V100 – ComputeCPP
 - 2 incorrect results, 2 miscompiled kernels
- Everything else was exact match

PERFORMANCE

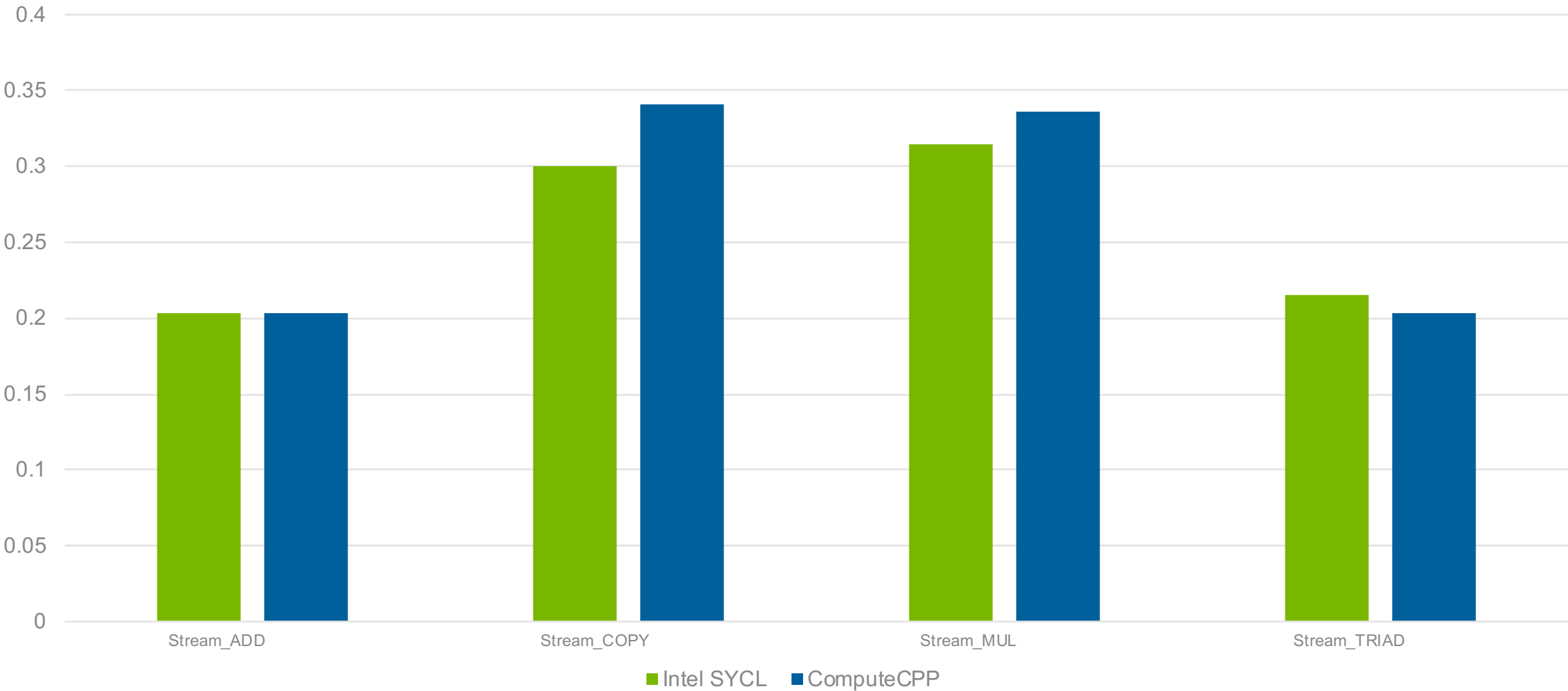


U.S. DEPARTMENT OF
ENERGY

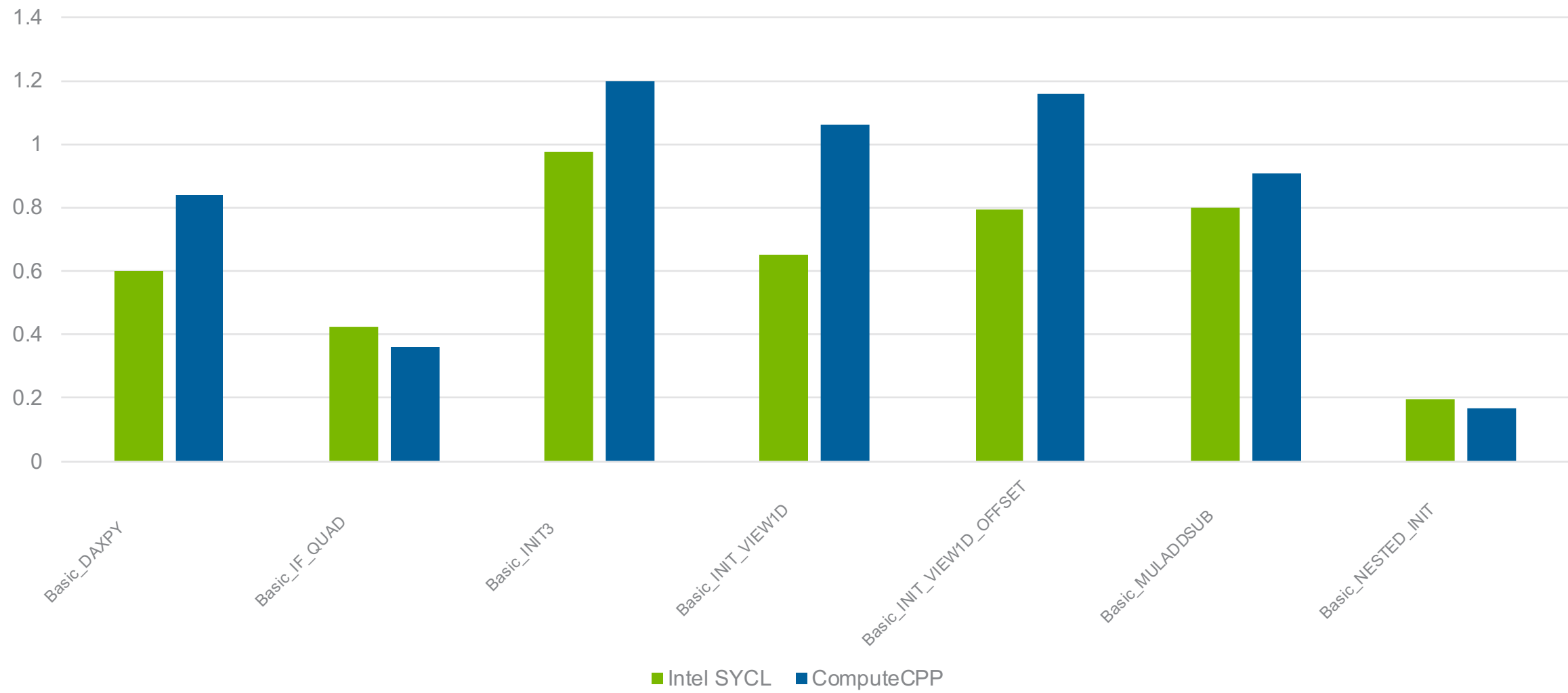
Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.



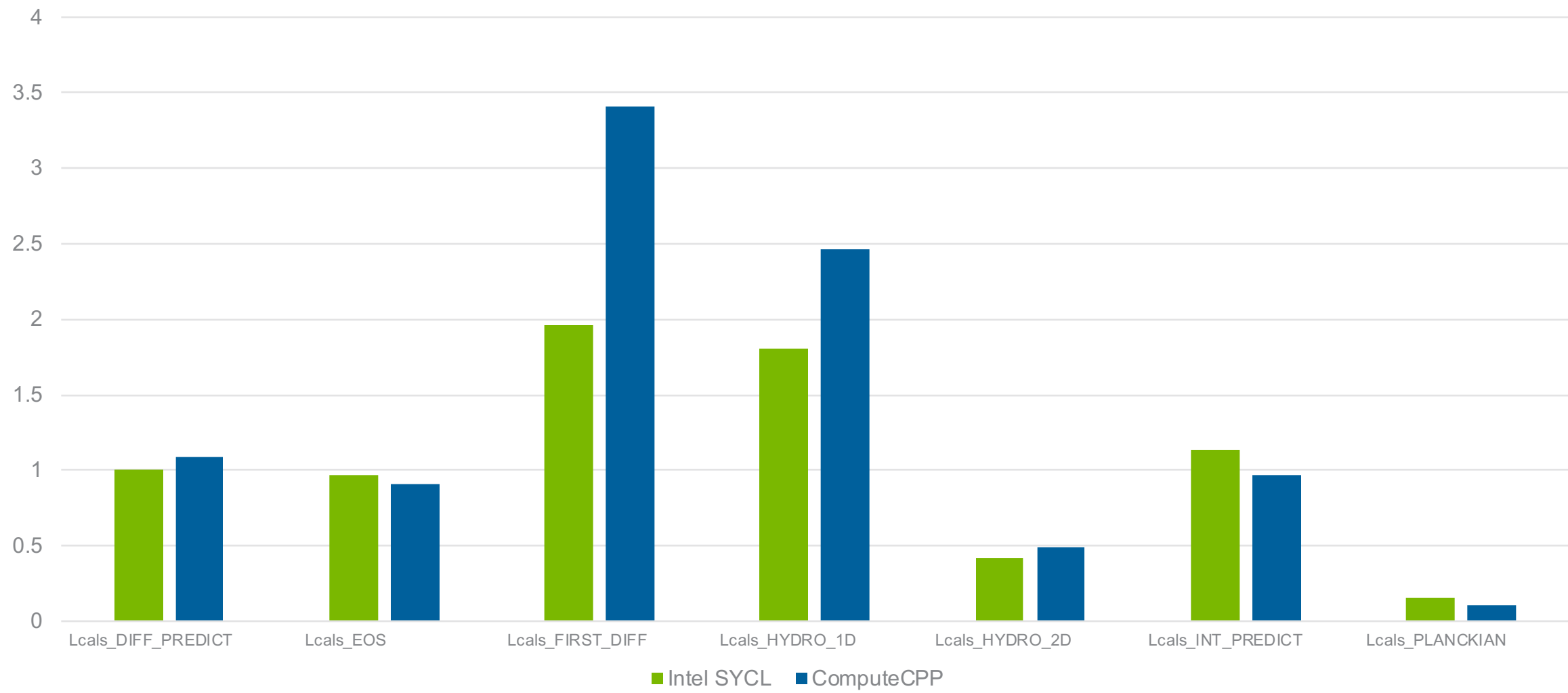
SKX – STREAM GROUP (SEC)



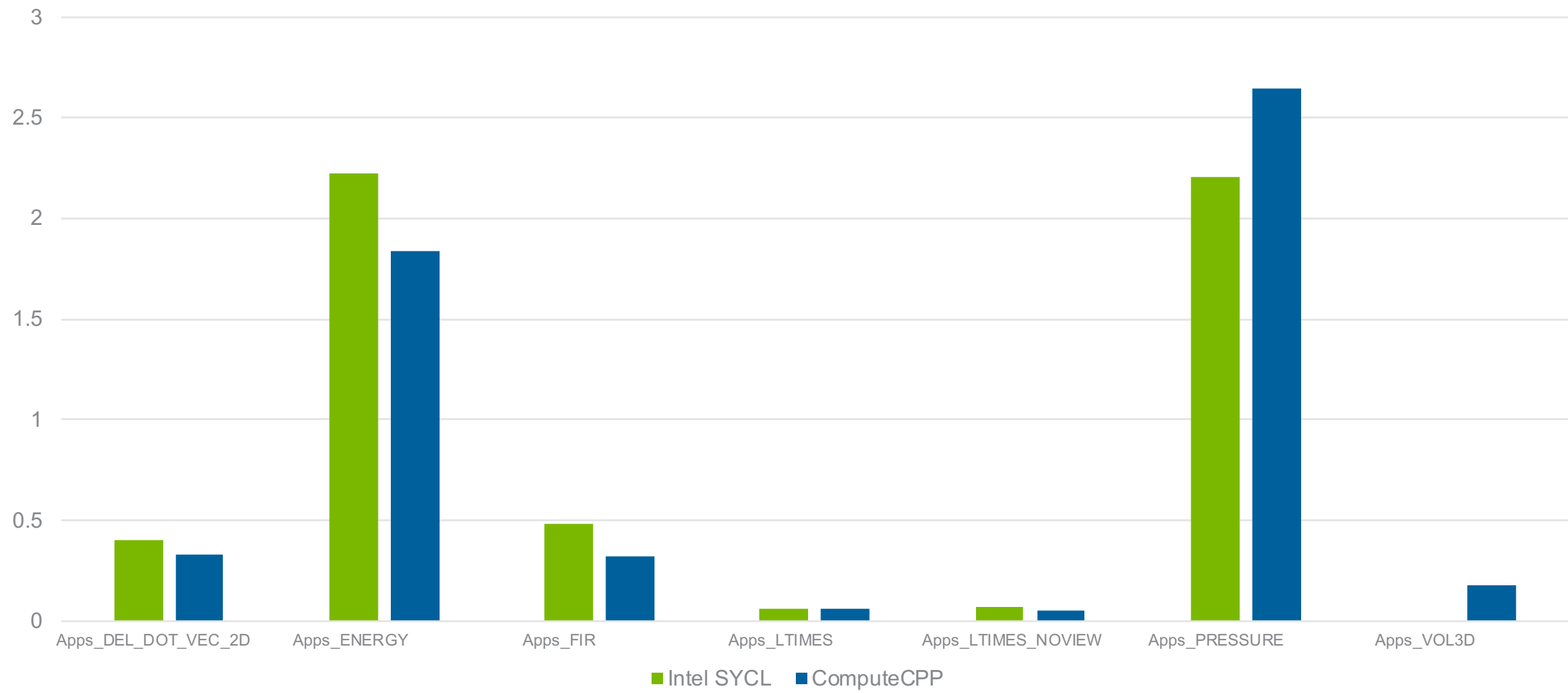
SKX – BASIC GROUP (SEC)



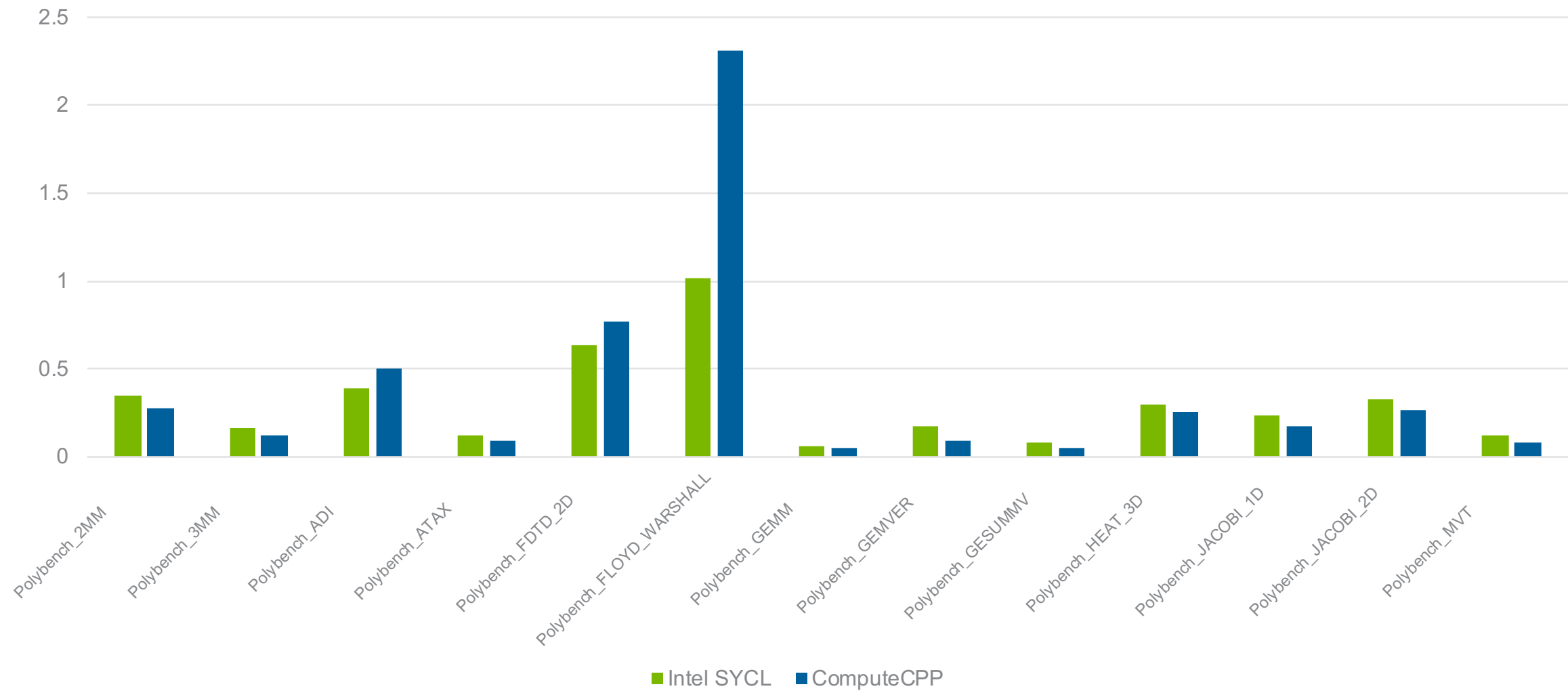
SKX – LCALS GROUP (SEC)



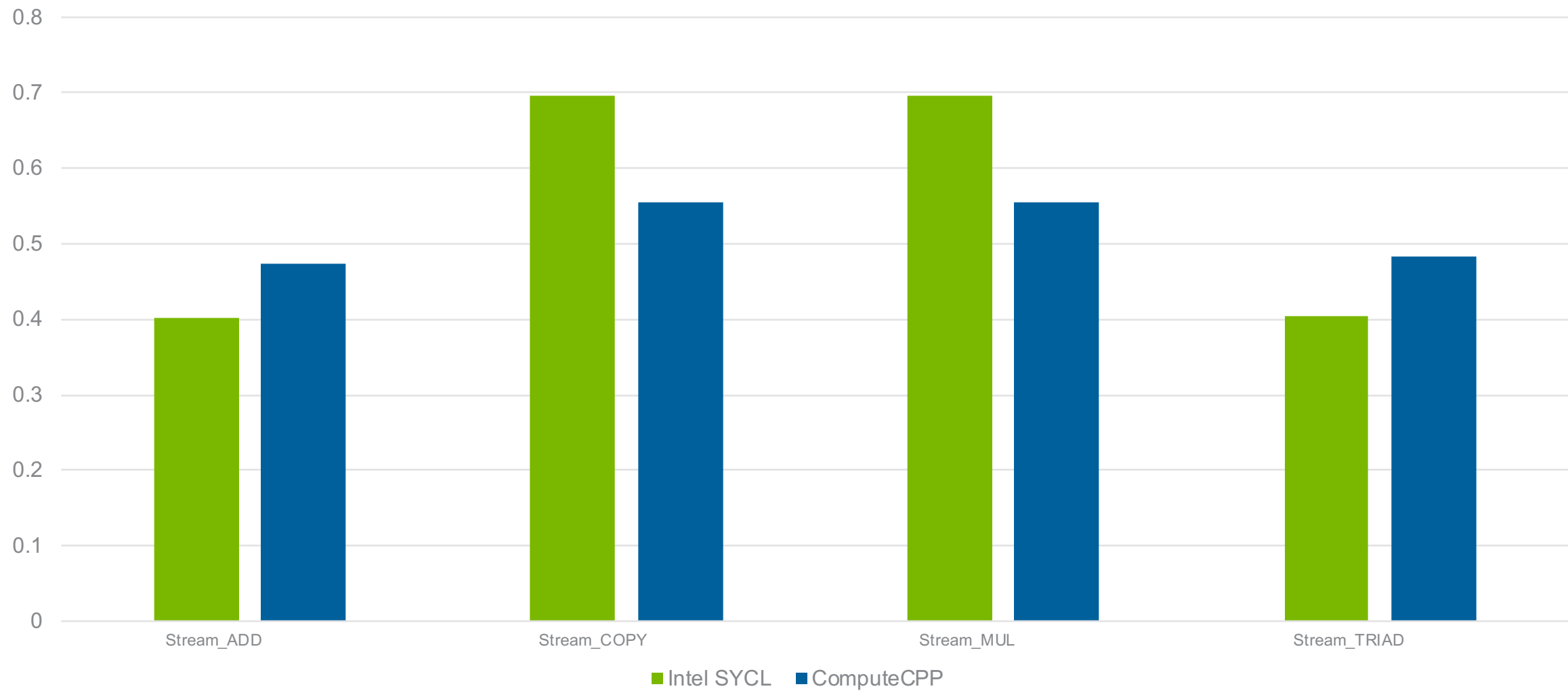
SKX – APPS GROUP (SEC)



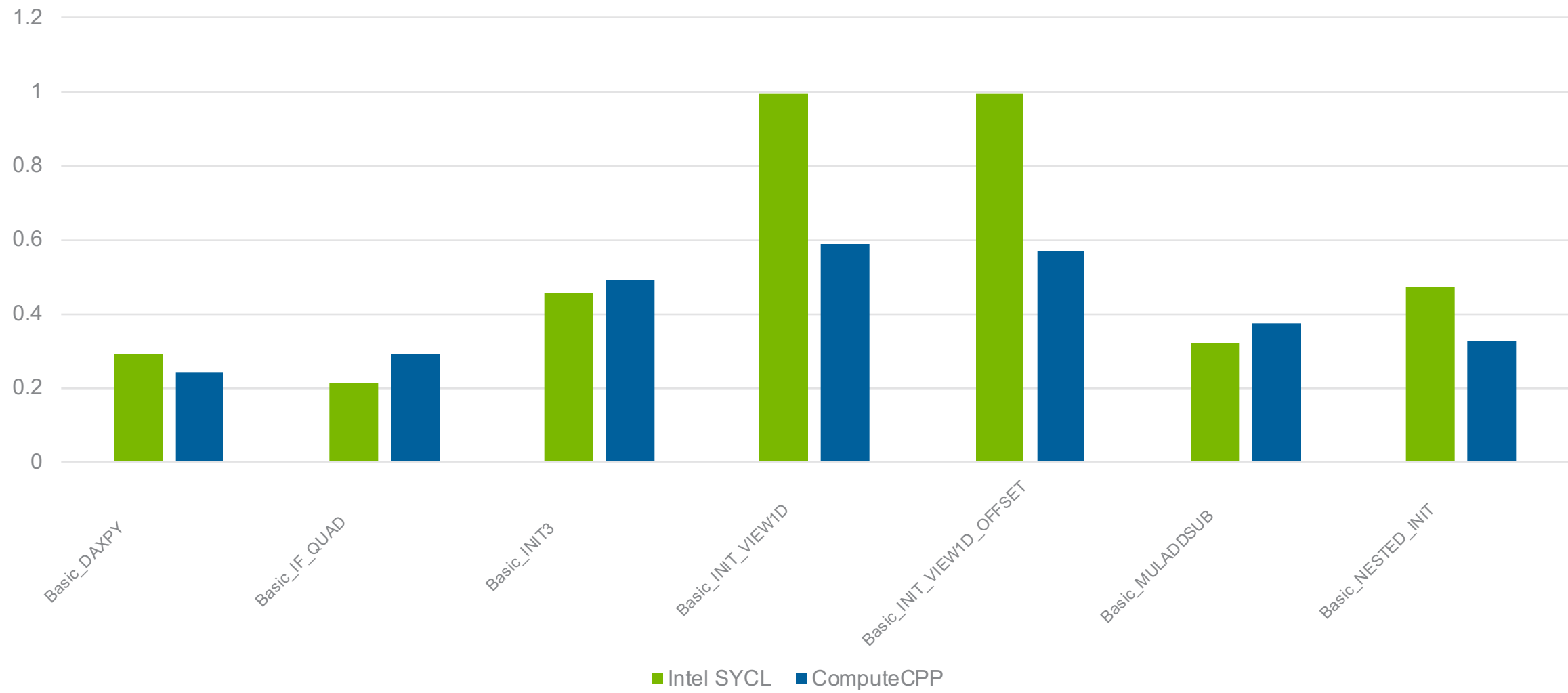
SKX – POLYBENCH GROUP (SEC)



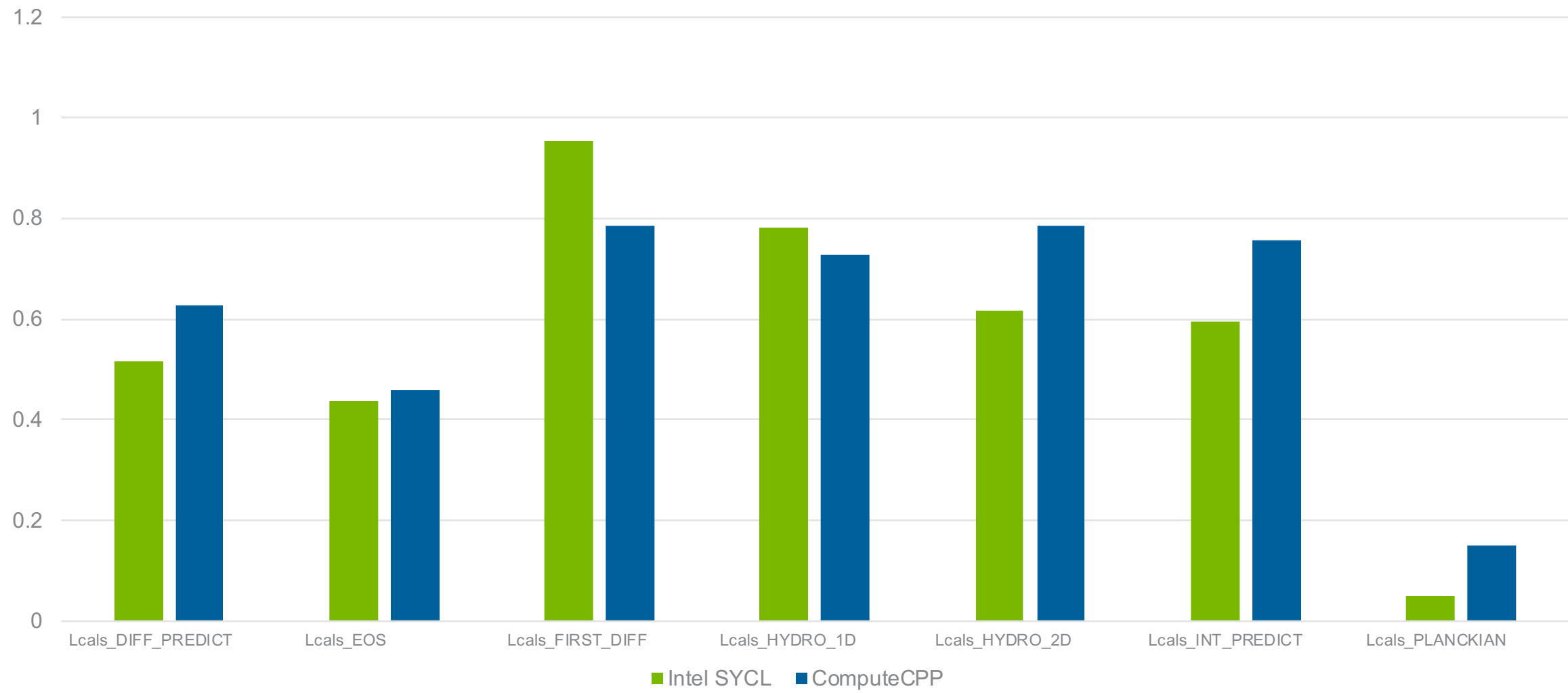
GEN9 – STREAM GROUP (SEC)



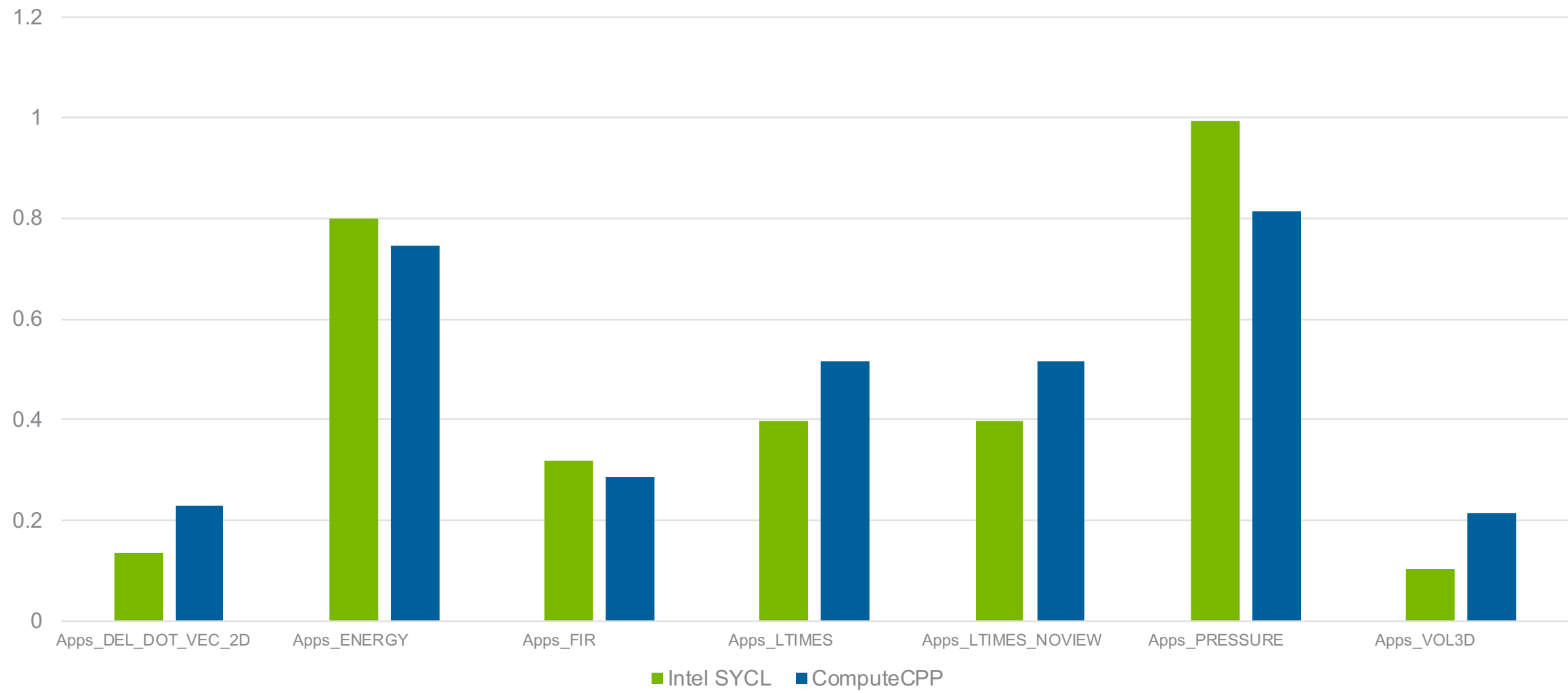
GEN9 – BASIC GROUP (SEC)



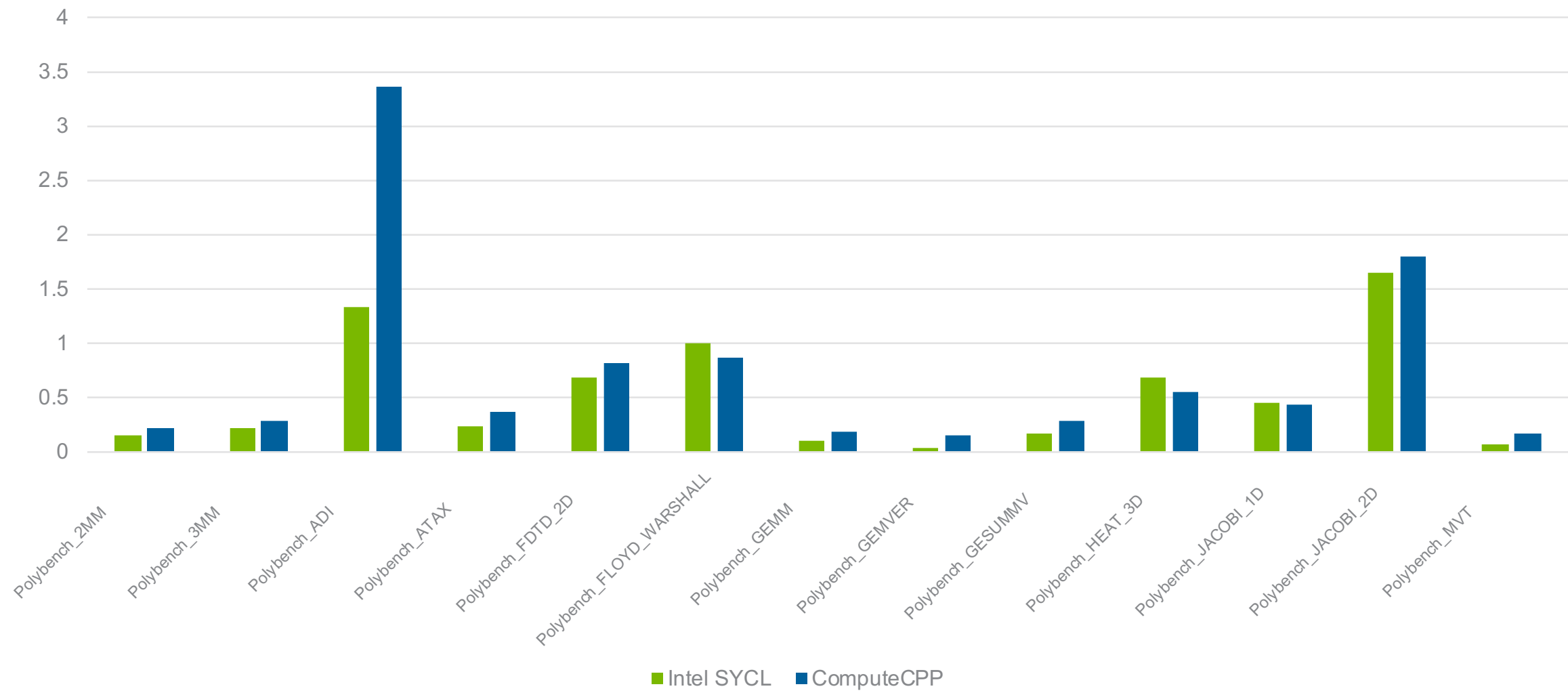
GEN9 – LCALS GROUP (SEC)



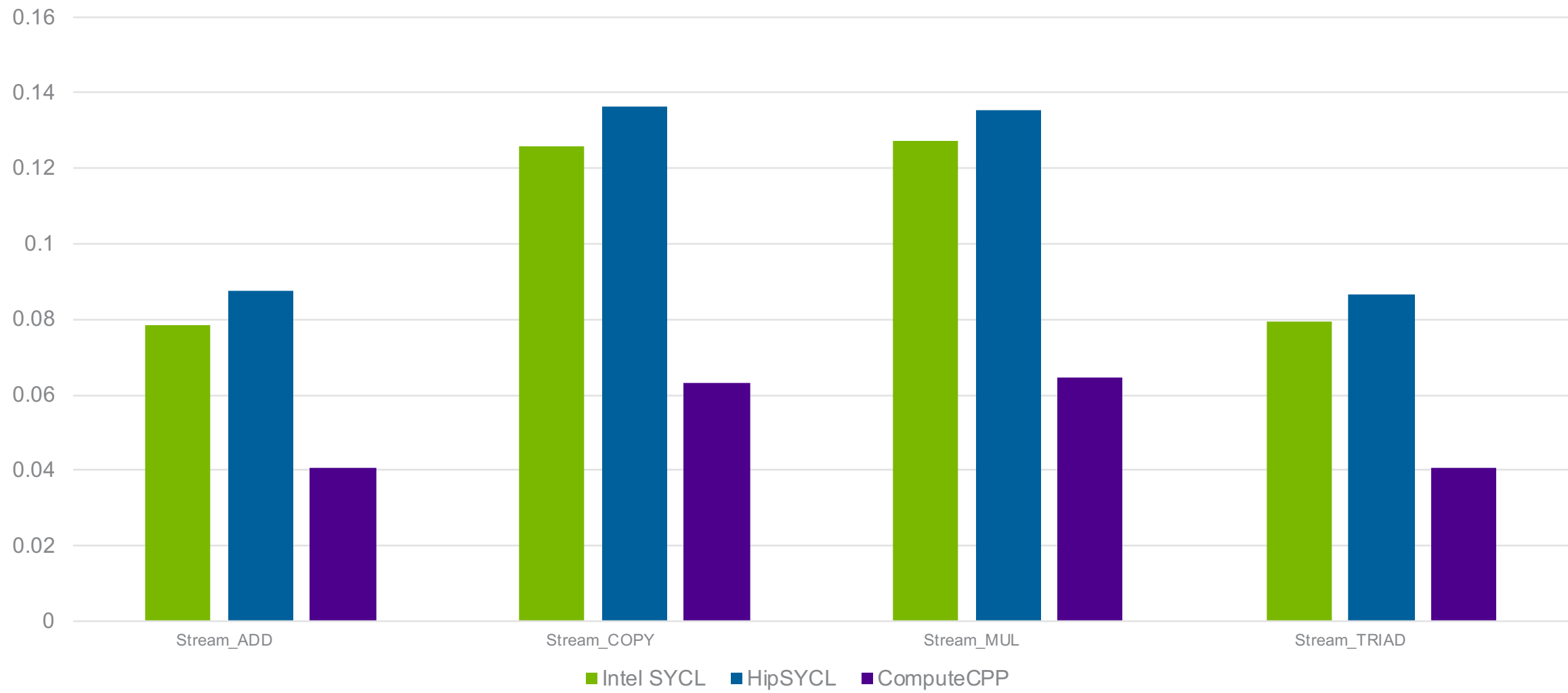
GEN9 – APPS GROUP (SEC)



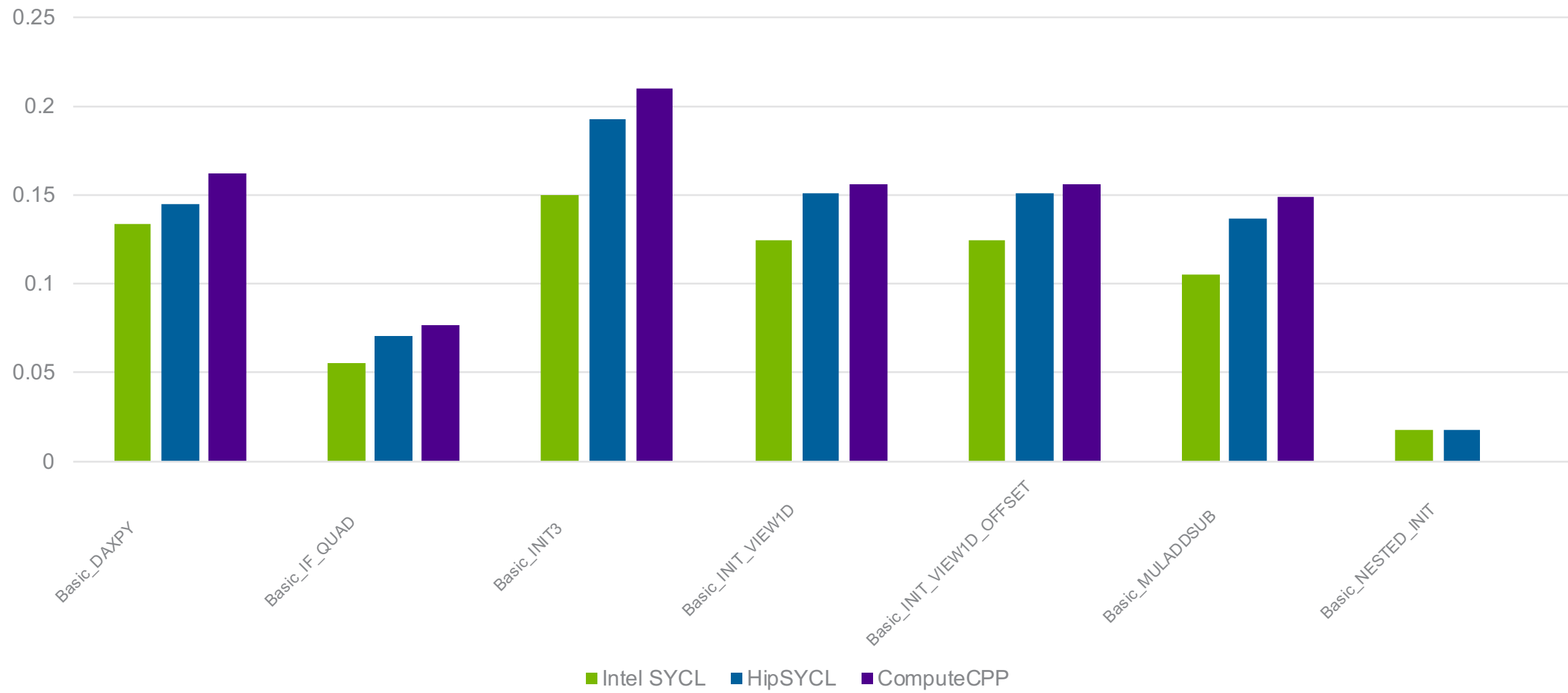
GEN9 – POLYBENCH GROUP (SEC)



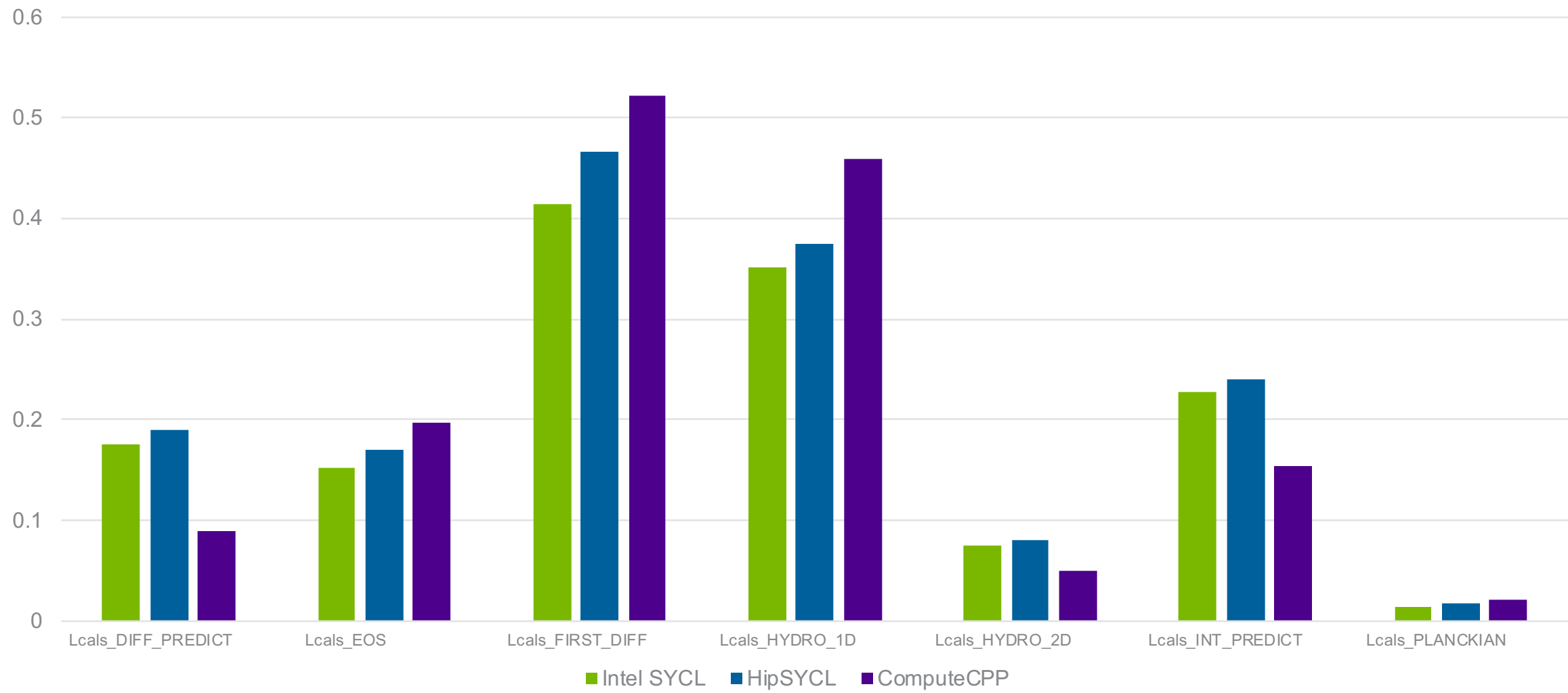
V100 – STREAM GROUP (SEC)



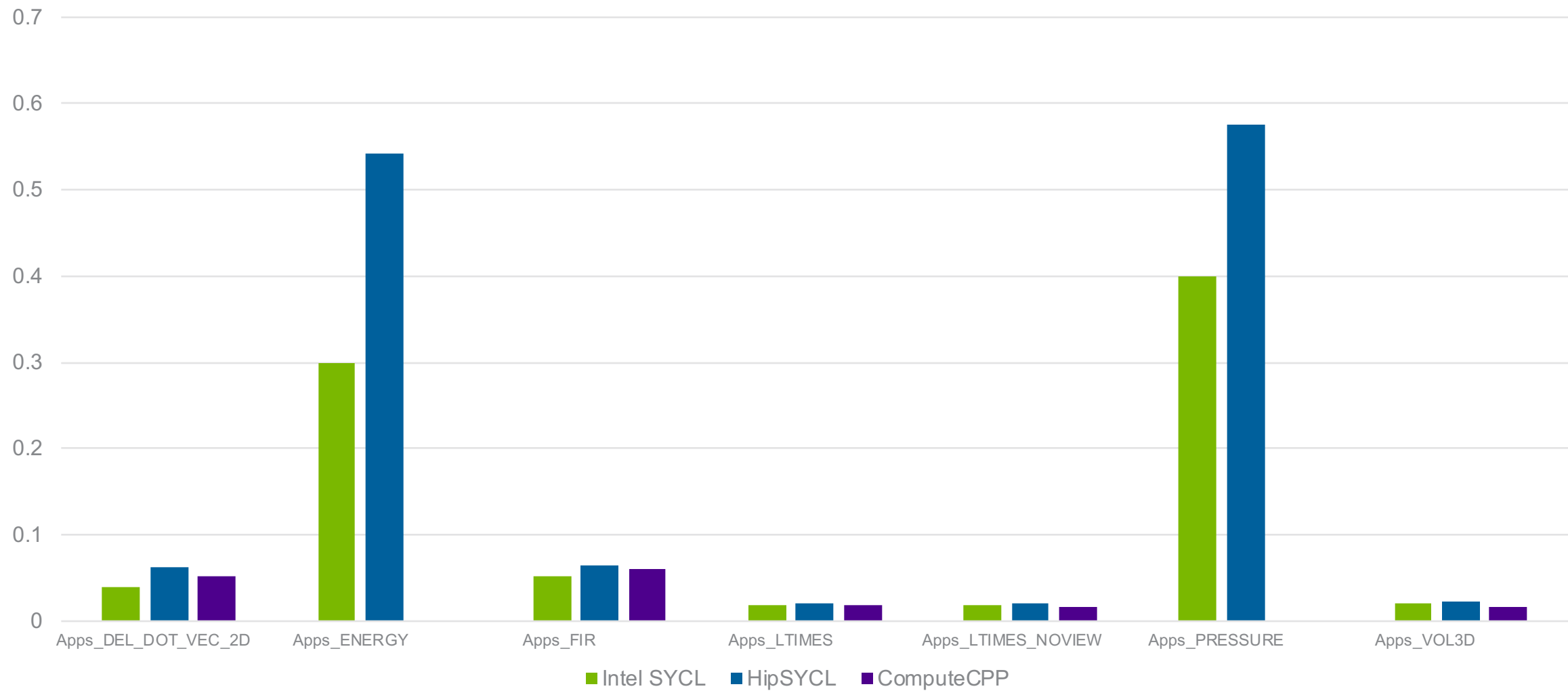
V100 – BASIC GROUP (SEC)



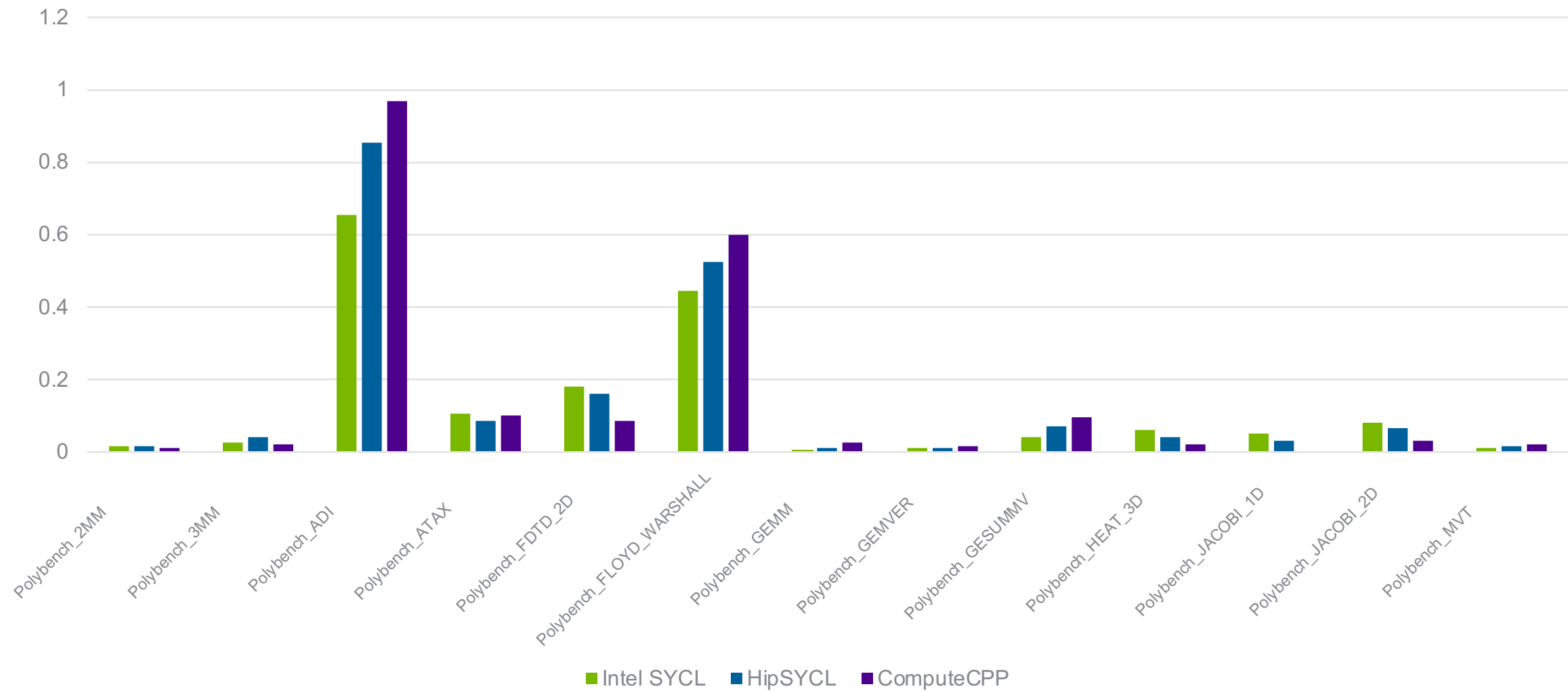
V100 – LCALS GROUP (SEC)



V100 – APPS GROUP (SEC)

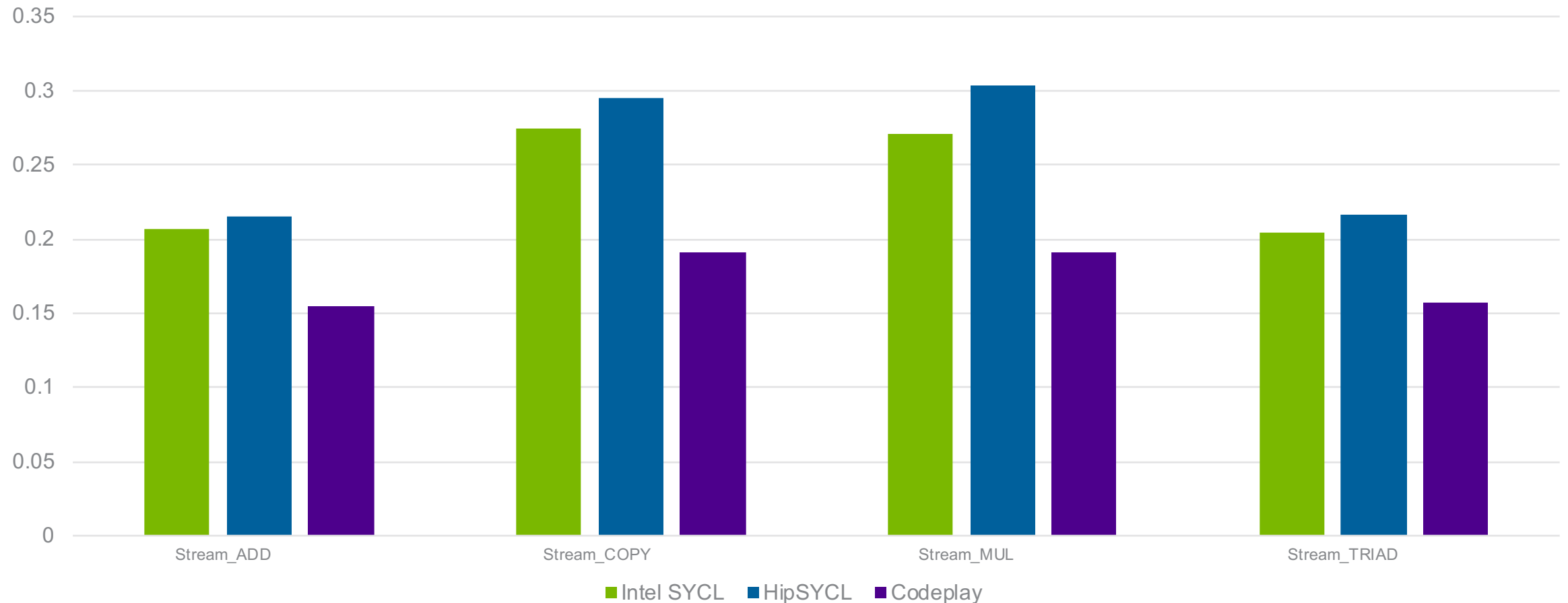


V100 – POLYBENCH GROUP (SEC)



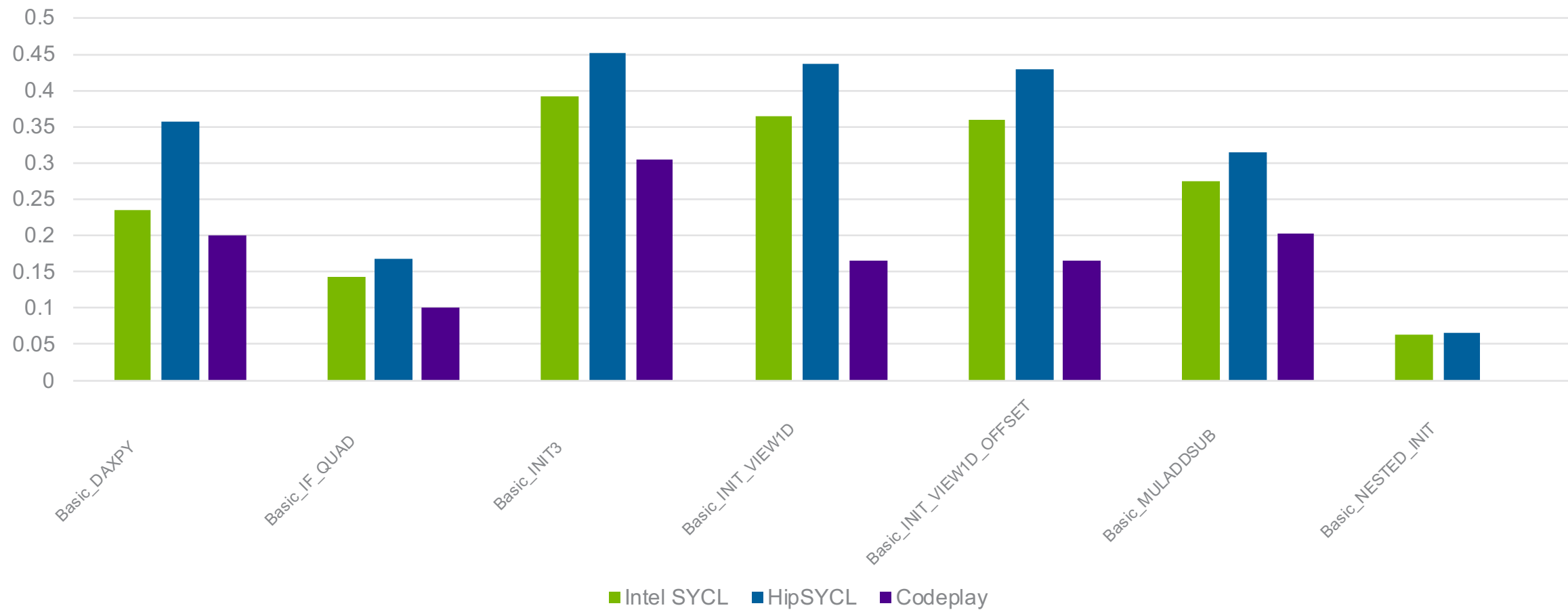
V100 – STREAM GROUP (SEC)

Size factor 5X



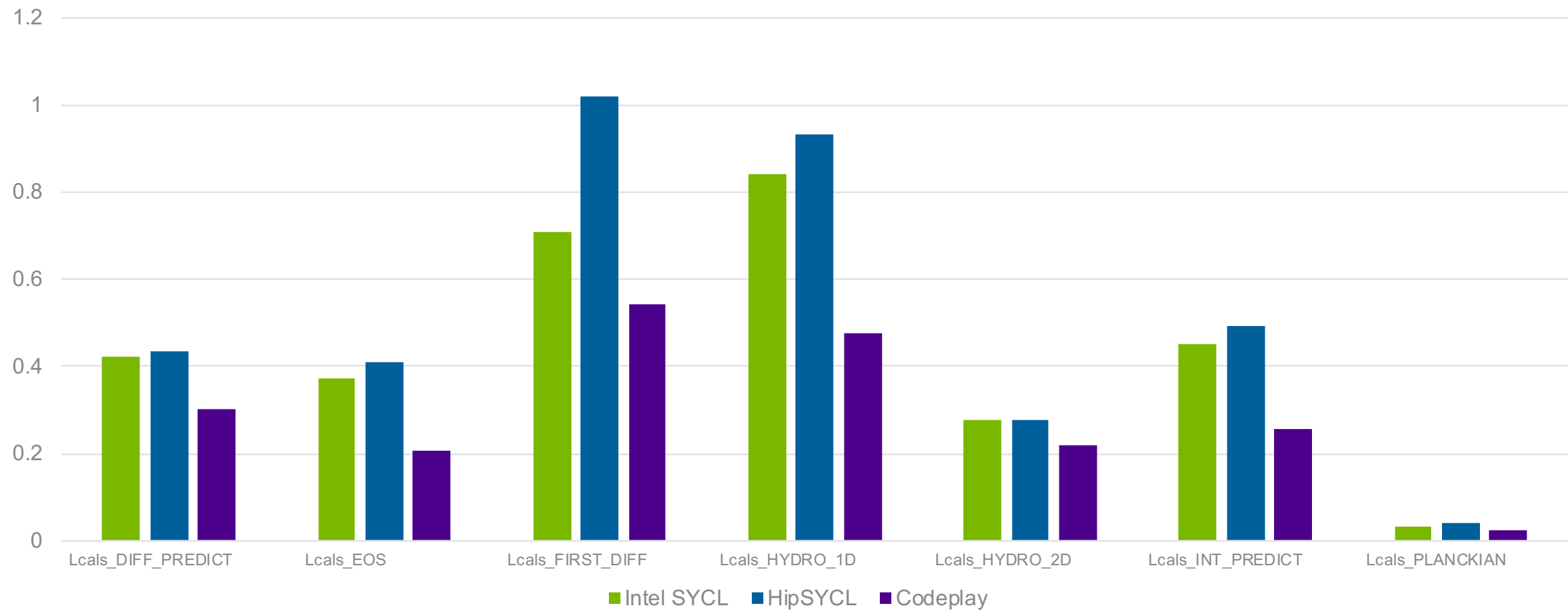
V100 – BASIC GROUP (SEC)

Size factor 5X



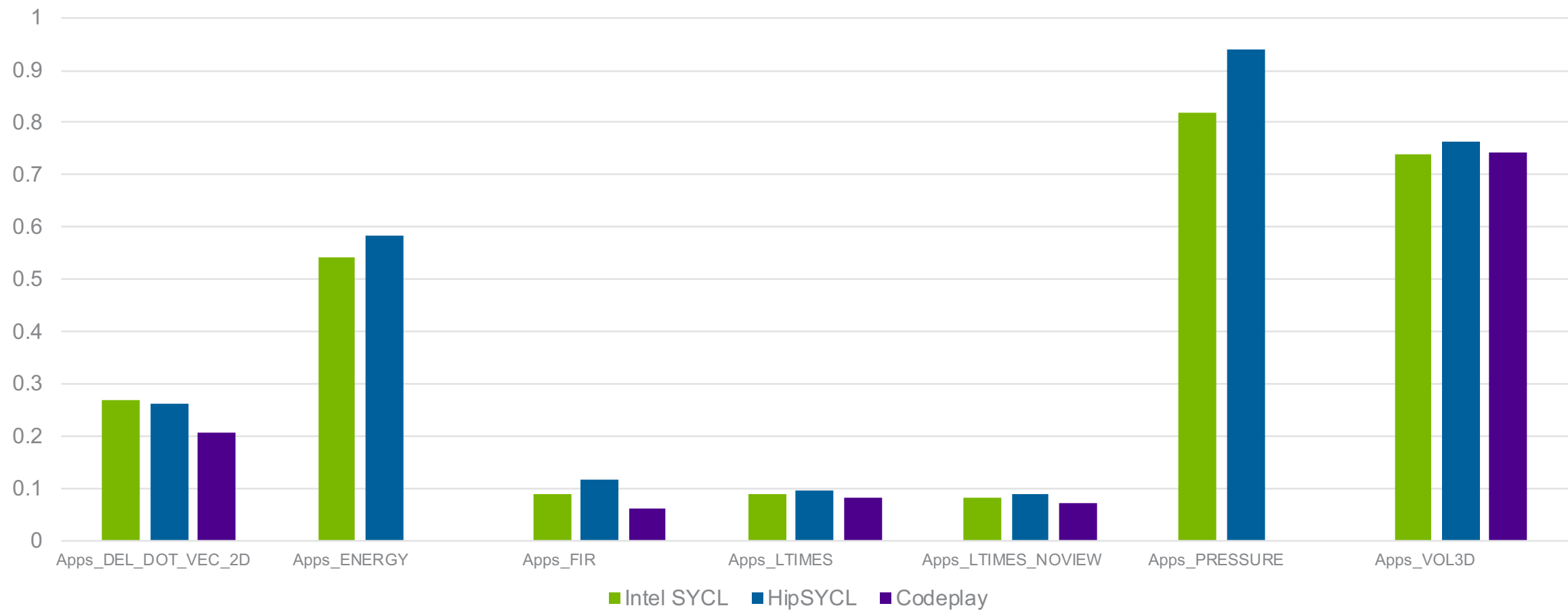
V100 – LCALS GROUP (SEC)

Size factor 5X



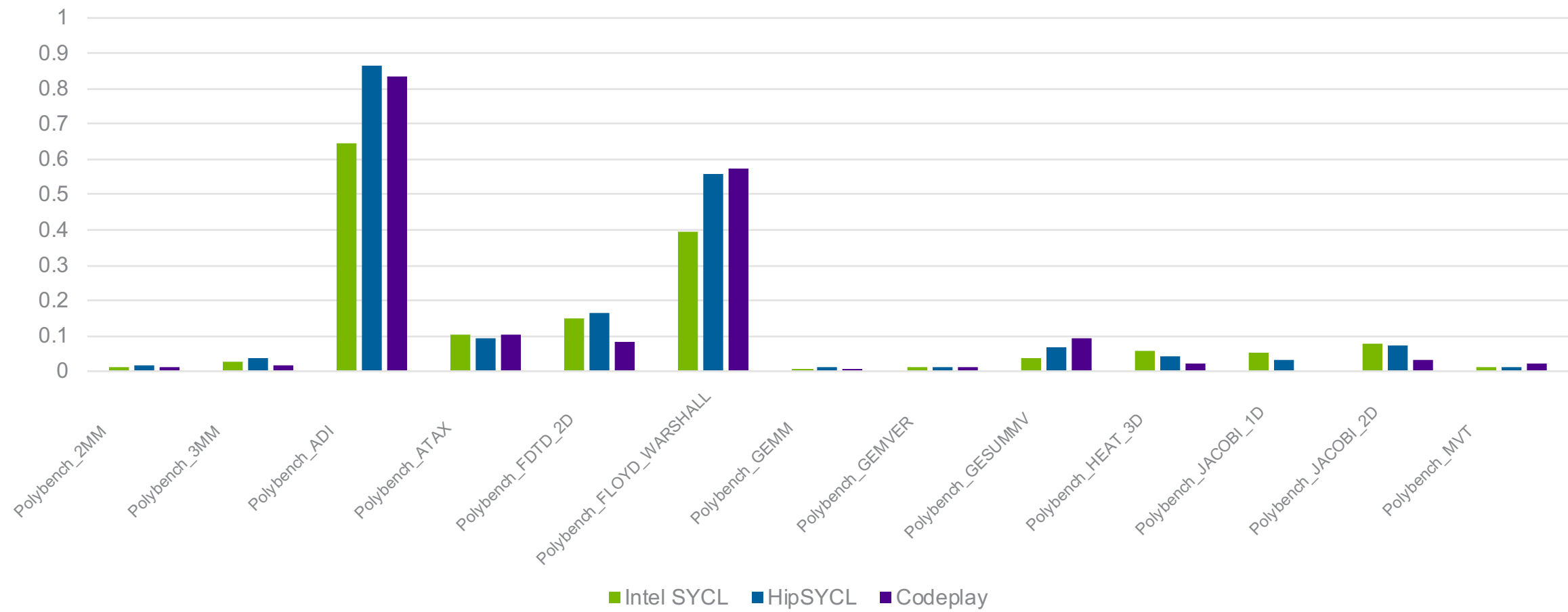
V100 – APPS GROUP (SEC)

Size factor 5X



V100 – POLYBENCH GROUP (SEC)

Size factor 5X

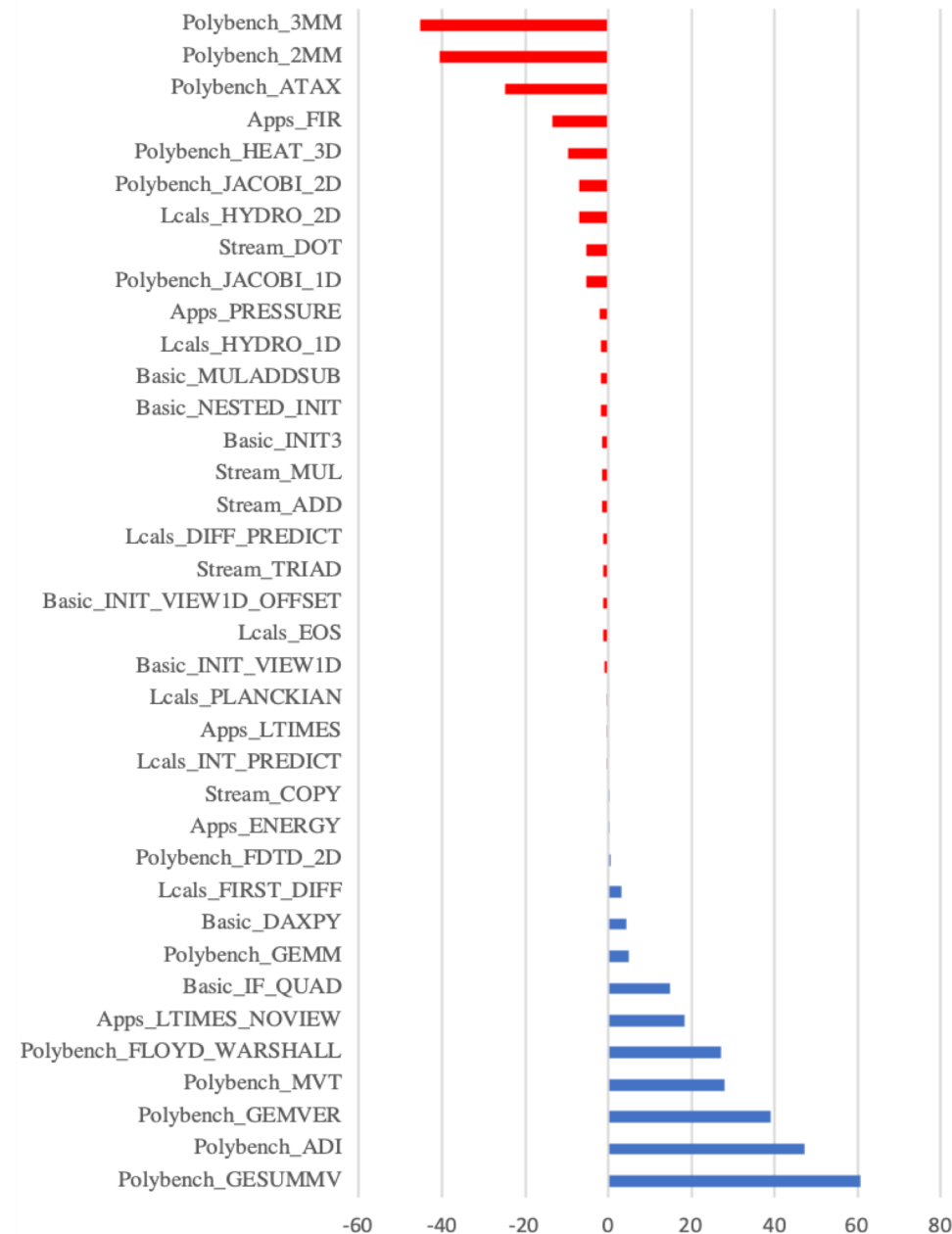


PREVIOUS WORK

SYCLcon 2020

“Evaluating the Performance of the hipSYCL Toolchain for HPC Kernels on NVIDIA V100 GPUS” [7]

- Conclusion
 - SYCL using hipSYCL is showing competitive performance to CUDA on NVIDIA devices
- Percent speedup of SYCL variant relative to the CUDA variant for kernel timings using nvprof



CONCLUSIONS

- Good ecosystem
 - Multiple compilers for each device
- Portable code
 - Minor feature support issues
- Performance is good across compilers
- More variance in compiler performance as complexity increases
 - Good to be able to test performance with various compilers

ACKNOWLEDGEMENTS

- ALCF, ANL and DOE
- ALCF is supported by DOE/SC under contract DE-AC02-06CH11357
- This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.
- We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

REFERENCES

- [1] Khronos OpenCL Working Group SYCL subgroup. 2018. SYCL Specification.
- [2] Richard D. Hornung and Holger E. Hones. 2020. RAJA Performance Suite.
<https://github.com/LLNL/RAJAPerf>
- [3] Intel SYCL. <https://github.com/intel/llvm/tree/sycl>.
- [4] Aksel Alpay and Vincent Heuveline. 2020. *SYCL beyond OpenCL: The architecture, current state and future direction of hipSYCL. In Proceedings of the International Workshop on OpenCL (IWOCCL '20). Association for Computing Machinery, New York, NY, USA, Article 8, 1.*
DOI:<https://doi.org/10.1145/3388333.3388658>
- [5] Codeplay. ComputeCPP. <https://developer.codeplay.com/products/computecpp/ce/home/>
- [6] C. Bertoni et al., "Performance Portability Evaluation of OpenCL Benchmarks across Intel and NVIDIA Platforms," *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, New Orleans, LA, USA, 2020, pp. 330-339,
DOI:<https://doi.org/10.1109/IPDPSW50202.2020.00067>
- [7] Brian Homerding and John Tramm. 2020. *Evaluating the Performance of the hipSYCL Toolchain for HPC Kernels on NVIDIA V100 GPUs. In Proceedings of the International Workshop on OpenCL (IWOCCL '20). Association for Computing Machinery, New York, NY, USA, Article 16, 1–7.*
DOI:<https://doi.org/10.1145/3388333.3388660>

THANK YOU



U.S. DEPARTMENT OF
ENERGY

Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

